ACSL TurtleBot3 e-Manual

June 2020

Contents

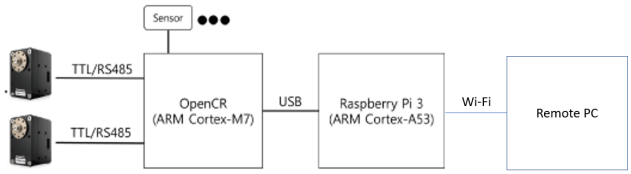
Abstract

This guide documents the changes made to TurtleBot3 software to meet the requirements of ACSL research group. It contains instructions on setting up the robot and code documentation.

1 Overview and Setup

TurtleBot3 is a small ROS-based mobile robot. The TurtleBot3 can be customized by changing the source code and addition of new hardware. TurtleBot3 Waffle Pi is equipped with a Raspberry Pi Camera Module (v2), a 360-degree Laser Distance Sensor (LDS) and Dynamixel XM430 210-T motors.

TurtleBot3 architecture is described in the following image.



The four major components of TurtleBot3 are

1. Raspberry Pi 3B+ Single Board Computer (SBC).
2. OpenCR Embedded System Board.
3. LDS and Camera Sensors.
4. Dynamixel Actuators.

Raspberry Pi has a WiFi module built onto the board which connects to a computer, known as the Remote PC. Remote PC is used to send commands for SLAM and Navigation to TB3 and visualization of the sensor data is also performed on the remote PC.

There is a one time setup that needs to be performed on the Remote PC, Raspberry Pi and OpenCR board. Most of the software is located in the form of a binary file that is burnt to the EEPROM of the STM32F746 chip on the OpenCR board. As the software used for ACSL projects is custom, it is burnt using Arduino IDE.

1.1 Remote PC Setup

To begin you will need to install the appropriate ROS version on your remote PC. The remote PC will be running ROS Melodic (EOL date: May 2023). The single board computer on Turtlebot3 will be running ROS Kinetic (EOL date: April 2021) for the ease of setup since ROS Melodic requires manual compilation when using Rasbian. No problem has been found on the communication between these 2 releases.

The following page from ros.org describes the process for adding and installing all necessary packages to your ROS Environment.

`http://wiki.ros.org/melodic/Installation/Ubuntu`

The next step is to configure network settings for permanent use. The Remote PC will be controlling the turtlebot pc via wifi using SSH protocol. Steps for configuring the wifi on the remote pc can be found here.

`https://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/#network-configuration`

For this project we will use the Linksys Router on the **Linksys04294** network. It is helpful to write down the IP address of the remote pc, since you will need it later to setup the turtlebot pc network settings.

1.2 Rasberry Pi SBC Setup

Now it is time to look at the turtlebot SBC. For the turtlebot 3 Waffle Pi model, the on board PC is a raspberry PI. This step requires a micro SD card with adapter, a monitor with an HDMI input, a USB keyboard and mouse, and a power source for the turtlebot. ROBOTIS provides a prebuilt desktop environment for the turtlebot with ROS kinetic. Instructions for flashing this distribution can be found here (**Step 6.2.1.2 Install Linux Based on Raspbian, Do not use the other two methods.**).

`https://emanual.robotis.com/docs/en/platform/turtlebot3/raspberry_pi_3_setup/#raspberry-pi-3-setup`

To use the Raspberry PI without a HDMI connection, use SSH. Example SSH command:

```
ssh pi@RASPBERRY_PI_IP
```

For the ease of use, VNC configuration is recommended. This provide remote desktop funtionality. In case of Debian Stretch (the version that the manual will use), install Real VNC by running this command on the Raspberry PI:

```
sudo apt update
sudo apt install realvnc-vnc-server realvnc-vnc-viewer
```

Enable VNC server by

```
sudo raspi-config
```

Then navigate to Interfacing Options. Scroll down and select VNC - Yes.

Ubuntu Desktop 18.04 comes with VNC client. On Windows, use VNC viewer `https://www.realvnc.com/en/connect/download/viewer/`

1.3 OpenCR Setup

The original source code for TurtleBot3 used the inbuilt PID controller of the Dynamixel XM430 210-T actuators. However, to benchmark the performance

of the controllers, the control is shifted from the motors to higher level software such as Simulink or a custom ROS node.

The source code for OpenCR can be downloaded by cloning the following repository.

`https://github.com/hanyiabc/ASCL_turtlebot3.git`

Arduino IDE can be used to burn the turtlebot3_core.ino file located in ACSL_turtlebot3/src/turtlebot3_core directory. Instructions for setting up the Arduino IDE for TurtleBot3 can be found in the following link.

`https://emanual.robotis.com/docs/en/parts/controller/opencr10/#arduino-ide`

2 Repository Directory Structure Introduction

The repository is a catkin workspace. A catkin workspace has a src folder that contains all the packages. The worth mentioning packages are:

- hls_lfcd_lds_driver
- raspicam_node
- ros_control
- simulink
- turtlebot3
  - turtlebot3_bringup
  - turtlebot3_navigation
  - turtlebot3_slam
  - turtlebot3_teleop
- turtlebot3_core
- turtlebot3_setup_motor
- turtlebot3_simulations
  - turtlebot3_gazebo

Not all packages are catkin package. simulink, turtlebot3_core turtlebot3_setup_motor are all just directories. All the rest are properly configured as catkin packages

---

hls_lfcd_lds_driver and raspicam_node are drivers for the PI Camera and lidar. These will be running in the single board computer.

ros_control contains all the configuration and nodes necessary to control the robot. It contains the PID controller configurationl, PID gains, etc. Two launch files are included for controlling the robot in either the simulations or in the physical world. They are:

- turtlebot3_control.launch
- turtlebot3_control_simulation.launch

These launch file are not ran directly, instead they are inlcuded by other launch file that we will introduce later. The PID gains can be tuned by either changing the parameters in the launch file or use a ros package called rqt_reconfiture to adjust the slider at runtime. There are 2 nodes under the ros_control package. They are differential_driver.py and ros_control_node. The differential_driver.py is a Python ros node that split the incoming cmd_vel topics into left and right velocity for the differential drive robot. The cmd_vel is the standard topic that ROS nav stack used to output the velocity command. It commands the robot in terms of translational velocity and orientation. The ros_control_node is a C++ ros node compiled from the soruce file: message_redirect.cpp. This node simply takes the left and right wheel velocities from joint_states and publish them into 2 Float64 messages for the PID controller.

The directory simulink contains all the MATLAB scrips and Simulink models for controlling the robot with MATLAB

turtlebot3_bringup contains all the launch file for bringing up the turtlebot3 in the SBC and the remote PC. turtlebot3_physical_nav_control.launch was added to the bringup package to provide a convinient one-file launch that will handles everything. This launch file brings up the turtlebot on the remote PC, runs the velocity control and the navigation stack with SLAM. A simulation version will be provided in the future.

turtlebot3_description contains the 3D model and URDF descripion of the turtlebot.

For Waffle Pi, turtlebot3_waffle_pi.gazebo.xacro and turtlebot3_waffle_pi.urdf.xacro are provided by the turtlebot package, however, for effort control, we added 2 more files. They are: turtlebot3_waffle_pi_effort_controller.gazebo.xacro and turtlebot3_waffle_pi_effort_controller.urdf.xacro These files are derived from the original descriptions and added ability to control effort in both the simulation and the physical robot.

turtlebot3_navigation contains ——

3 Torque Control using Dynamixel

Dynamixel is a microcontroller based actuator with in-built PID controller. OpenCR communicates with Dynamixel using packet communication via TTL/RS45 ports.

Hardware level abstraction is achieved via Dynamixel SDK library available in several programming languages. (CPP used in our case.) Dynamixel register addresses and byte sizes for both RAM and EEPROM memory provided in control table. RAM is most frequently used memory for robot applications, while startup settings are stored on EEPROM.

The Control Table is a structure that consists of multiple Data fields to store status or to control the device. Users can check current status of the device by reading a specific Data from the Control Table with Read Instruction Packets. WRITE Instruction Packets enable users to control the device by changing specific Data in the Control Table. Packet sizes range from $1 - 4$ bytes.

Following is a snapshot of the Dynamixel EEPROM Control Table. Each data in the Control Table is restored to initial values when the device is turned on. Default values in the EEPROM area (addresses 0-63) are initial values of the device (factory default settings). If any values in the EEPROM area are modified by a user, modified values will be restored as initial values when the device is turned on. Initial Values in the RAM area are restored when the device is turned on.

Address	Size (Byte)	Data Name	Access	Default Value	Range	Unit
0	2	Model Number	R	1,030	-	-
2	4	Model Information	R	-	-	-
6	1	Firmware Version	R	-	-	-
7	1	ID	RW	1	0 ~ 253	-
8	1	Baud Rate	RW	1	0 ~ 7	-
9	1	Return Delay Time	RW	250	0 ~ 254	2 [µsec]
10	1	Drive Mode	RW	0	0 ~ 1	-
11	1	Operating Mode	RW	3	0 ~ 16	-
12	1	Secondary(Shadow) ID	RW	255	0 ~ 252	-
13	1	Protocol Type	RW	2	1 ~ 2	-
20	4	Homing Offset	RW	0	-1,044,479 ~ 1,044,479	1 [pulse]
24	4	Moving Threshold	RW	10	0 ~ 1,023	0.229 [rev/min]
31	1	Temperature Limit	RW	80	0 ~ 100	1 [°C]
32	2	Max Voltage Limit	RW	160	95 ~ 160	0.1 [V]
34	2	Min Voltage Limit	RW	95	95 ~ 160	0.1 [V]
36	2	PWM Limit	RW	885	0 ~ 885	0.113 [%]
38	2	Current Limit	RW	1,193	0 ~ 1,193	2.69 [mA]
44	4	Velocity Limit	RW	330	0 ~ 1,023	0.229 [rev/min]

For the purpose of torque control, the operating mode of the dynamixel motor needs to be set to **MODE 0**. This can be done using the motor setup code in the git repository.

Once the operating mode is set. The OpenCR code is changed to write torque values to the Dynamixel RAM addresses, when the robot is live. The addresses that are useful for this purpose are given in the image below.

102	2	Goal Current	RW	-	-Current Limit(38) ~ Current Limit(38)	2.69 [mA]
104	4	Goal Velocity	RW	-	-Velocity Limit(44) ~ Velocity Limit(44)	0.229 [rev/min]
108	4	Profile Acceleration	RW	0	0 ~ 32,767 0 ~ 32,737	214.577 [rev/min^2] 1 [ms]
112	4	Profile Velocity	RW	0	0 ~ 32,767	0.229 [rev/min]
116	4	Goal Position	RW	-	Min Position Limit(52) ~ Max Position Limit(48)	1 [pulse]
120	2	Realtime Tick	R	-	0 ~ 32,767	1 [msec]
122	1	Moving	R	0	0 ~ 1	-
123	1	Moving Status	R	0	-	-
124	2	Present PWM	R	-	-	-
126	2	Present Current	R	-	-	2.69 [mA]
128	4	Present Velocity	R	-	-	0.229 [rev/min]
132	4	Present Position	R	-	-	1 [pulse]

The following method is added to the TurtleBot3MotorDriver class, which takes in input torque value and converts it into appropriate register values for writing to the appropriate addresses.

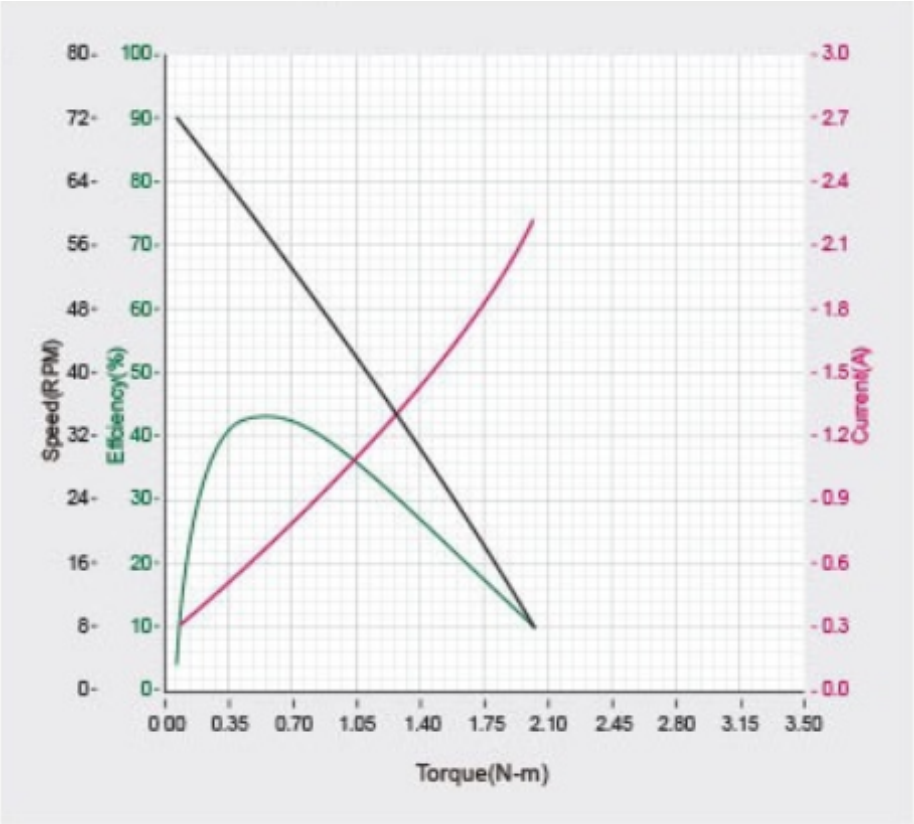
```
bool TurtleBot3MotorTorqueDriver::controlMotor(float *torque)
{
        bool dxl_comm_result = false;

        uint16_t wheel_current_cmd[2];
        wheel_current_cmd[LEFT] =
            CURRENT_TO_OUTPUT(TORQUE_TO_CURRENT(torque[LEFT]));
        wheel_current_cmd[RIGHT] =
            CURRENT_TO_OUTPUT(TORQUE_TO_CURRENT(torque[RIGHT]));

        wheel_current_cmd[LEFT] = constrain(wheel_current_cmd[LEFT],
            -dynamixel_limit_max_current_, dynamixel_limit_max_current_);
        wheel_current_cmd[RIGHT] = constrain(wheel_current_cmd[RIGHT],
            -dynamixel_limit_max_current_, dynamixel_limit_max_current_);

        dxl_comm_result = writeTorque((int16_t)wheel_current_cmd[LEFT],
            (int16_t)wheel_current_cmd[RIGHT]);
        if (dxl_comm_result == false)
```

```
            return false;

        return true;
}
```

CURRENT_TO_OUTPUT and TORQUE_TO_CURRENT are macro functions that does the conversions from torque to current and current to a 2 byte value that the microcontroller undertand. All the necessary macros are defined as

```
#define MAX_CURRENT_11V1   2.1
#define MAX_TORQUE_11V1    2.7

#define CURRENT_GOAL_UNIT  2.69 //in miliamps
#define TORQUE_TO_CURRENT(t) t * (MAX_CURRENT_11V1 / MAX_TORQUE_11V1)
// convert torque to current in amp
#define CURRENT_TO_OUTPUT(a) (uint16_t)(a * 1000 / CURRENT_GOAL_UNIT)
#define CURRENT_TO_TORQUE(t) t / (MAX_CURRENT_11V1 / MAX_TORQUE_11V1)
// convert current in amp to torque in N-m
#define OUTPUT_TO_CURRENT(a) a / 1000 * CURRENT_GOAL_UNIT
```

This conversion is based on a linear approximation of the actuator performance graph.



This function below takes the converted value and write to both motors using the Dynamixel SDK APIs.

```
bool TurtleBot3MotorTorqueDriver::writeTorque(int16_t left_value,
    int16_t right_value)
{
    bool dxl_addparam_result;
    int8_t dxl_comm_result;

    dxl_addparam_result =
        groupSyncWriteTorque_->addParam(left_wheel_id_, (uint8_t
        *)&left_value);
    if (dxl_addparam_result != true)
        return false;

    dxl_addparam_result =
        groupSyncWriteTorque_->addParam(right_wheel_id_, (uint8_t
        *)&right_value);
    if (dxl_addparam_result != true)
        return false;
```

```
    dxl_comm_result = groupSyncWriteTorque_->txPacket();
    if (dxl_comm_result != COMM_SUCCESS)
    {
        Serial.println(packetHandler_->getTxRxResult(dxl_comm_result));
        return false;
    }

    groupSyncWriteTorque_->clearParam();
    return true;
}
```

Subscribers are added to subscribe to the topics left_torque right_torque. The following use the function defined above and write the correspounding values to the correct memory location of the Dynamixel motor controller when new message comes in. When there is no new message, it will timeout and write zeros to both motors.

```
if ((t-tTime[0]) >= (1000 / CONTROL_MOTOR_TORQUE_FREQUENCY))
{
    updateGoalTorque();
    //this timeout will stop the motor if no message comes in
    if ((t-tTime[6]) > CONTROL_MOTOR_TIMEOUT)
    {
    motor_driver.controlMotor(zero_torque);
    }
    else {
    motor_driver.controlMotor(torque);
    }
    tTime[0] = t;
}
```

4 Custom controller in Simulink